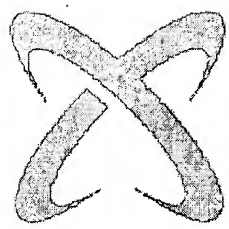


xpedior™

Products and Tools Group

Messaging Platform Overview

June 2000

Xpedior, Inc.
San Francisco, Los Angeles, Denver,
Boston, London(UK), Perth(Australia)
www.xpedior.com

Table of Contents

Preface

Audience and Purpose

Messaging Platform Overview is an introduction to Messaging Platform for both technical and non-technical readers. It provides both high-level and detailed descriptions of what Messaging Platform does and how it works. It also explains the benefits Messaging Platform provides and describes the tools, components, and frameworks that make up Messaging Platform. It also provides information about how to create a Messaging Platform component using the Assembler Component and how to use Messaging Platform monitoring tools.

Document Overview

Messaging Platform Overview contains five sections:

Introduction to Messaging Platform. This section explains what Messaging Platform is and how it works at a high level. It provides scenarios illustrating Messaging Platform's use. It also describes the tools, components, and frameworks that comprise Messaging Platform and discusses the benefits provided by Messaging Platform.

Inbound Message Process. This section describes the role of Messaging Platform in receiving requests from and external sender, processing the requests for use by the recipient, and returning a response from the recipient to the original sender.

Outbound Message Process. This section describes how Messaging Platform sends requests from a Component Platform component to an external service.

Messaging Platform Component Templates. This section describes the Messaging Platform templates that can be used to create Messaging Platform components.

Monitoring Tools. This section describes Messaging Platform monitoring tools.

Additional Information

This section contains URLs to Websites that provide additional information about some of the topics discussed in this document.

HTTP References

The Hypertext Transfer Protocol (RFC2068: Hypertext Transfer Protocol) is available at: http://info.internet.isi.edu/in-notes/rfc/files/rfc2068.txt

An addendum to the protocol is available at:
http://www.w3.org/Protocols/History.html

XML References

The XML Specification is available at: http://www.w3.org/TR/WD-xml-lang.html

The XML-Data Specification is available at:
http://www.w3.org/TR/1998/NOTE-XML-data/

Document Content Description for XML is available at:
http://www.w3.org/TR/NOTE-dcd.html

Information about Namespaces in XML is available at:
http://www.w3.org/TR/REC-xml-names/

Information about the XML Linking Language is available at:
http://www.w3.org/1999/07/WD-xlink-19990726.html

A description of XML Schema Part 1: Structures is available at:
http://www.w3.org/TR/1999/WD-xmlschema-1-19991105/

A description of XML Schema Part 2: Datatypes is available at:
http://www.w3.org/TR/1999/WD-xmlschema-2-19991105/

SOAP References

The Simple Object Access Protocol (SOAP) specification is available at:
http://msdn.microsoft.com/xml/general/soap_v09.asp

Introduction to Messaging Platform

This section explains what Messaging Platform is and how it works; provide some scenarios that illustrate its use; describe the tools, components, and interfaces available for it; and describe its features.

What Is Messaging Platform

Messaging Platform provides a mechanism for connecting heterogeneous systems within and across enterprises. Using Messaging Platform, you can make your enterprise services available over the Web. Your system can also use Messaging Platform to access the services of a different remote enterprise. Messaging Platform can be implemented within your enterprise, via an intranet, to provide communication between disparate systems. Finally, Messaging Platform provides the flexibility to work with proprietary systems, such as an ERP, through the use of adapters and connectors.

Messaging Platform uses standard Internet protocols to make cross-enterprise exchanges possible. Messaging Platform converts information from application-specific formats into XML documents, transports the documents between senders and recipients via a protocol (such as HTTP), repackages the XML documents for use by the recipient, and returns information from the recipient back to the sender. Messaging Platform implements Microsoft's Simple Object Access Protocol specification (SOAP), which describes a way to use HTTP as the base transport mechanism for exchanging XML documents. SOAP is an emerging Internet standard.

Note: Although SOAP the Messaging Platform's default transport mechanism, you can implement other mechanism in its place. Messaging Platform has been designed to allow flexibility in the adoption of standards and protocols.

How Does Messaging Platform Send and Receive Information?

The XML documents exchanged via Messaging Platform represent objects or, more precisely, contain the methods and parameters needed to invoke a function that can return specific information to a sender. In effect, Messaging Platform provides a way for an application to make its existing methods available as services that other applications can use. Likewise, it provides a way for an application to access the services offered by a remote application. Thus, with Messaging Platform, your existing enterprise is transformed into a Web-enabled enterprise.

At a high level, Messaging Platform performs three functions:

Generates and encodes XML documents from serialized objects. When an object is serialized, it is converted into a stream of bytes. Messaging Platform serializes objects and then packages them as XML documents that conform to the specific requirements of SOAP or a different encoding mechanism of your choice.

Transports XML documents. Messaging Platform uses HTTP to transport XML documents between a sender and a recipient. To ensure that XML documents are sent and received correctly, Messaging Platform establishes and stores connection information. When it receives an XML document from a sender, Messaging Platform makes sure that the intended recipient exists, and it then makes sure that the message is transmitted to the recipient correctly. Although HTTP is Messaging Platform's default transport protocol, other protocols can be used in its place.

Runs requested services. The XML documents created by Messaging Platform contain three types of information: the name of the service; the method to be run by that service; and the information, or parameters, to be given to these services. Messaging Platform takes these parameters from the XML documents and passes them to the requested service. Messaging Platform subsequently sends the information returned as a result of running the service to the requestor.

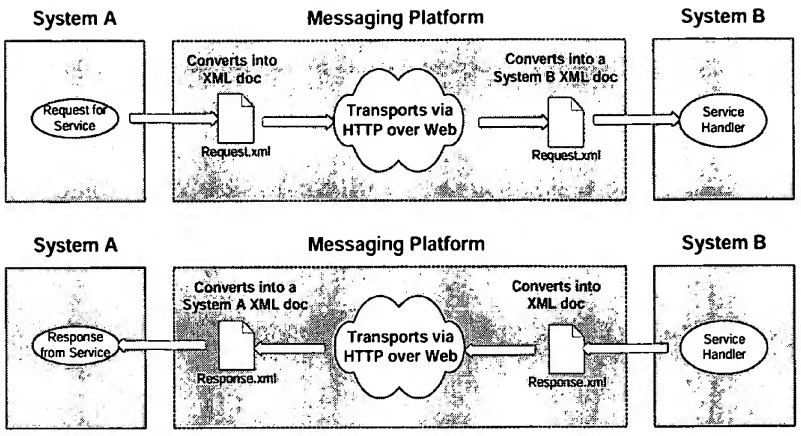


Figure 1. High-level illustration of Messaging Platform

The secret to Messaging Platform lies in its use of XML schemas. A schema is a document that presents a collection of rules for naming and defining objects, their properties, and their attributes. Two systems can use Messaging Platform to exchange object data packaged in XML documents if they use the same schema. As long as the XML documents exchanged between the systems are formatted in compliance with the schema's rules, the systems will be able to invoke each others' methods.

Messaging Platform Scenarios

This section describes some scenarios that illustrate how Messaging Platform can be implemented across or within enterprises. For these scenarios, we've invented a .com company called Car.com. Car.com provides services for car buyers. For example, buyers can purchase or lease a vehicle over the Web, arrange for its financing, and shop for and purchase for car insurance. To provide these interactive services, Car.com's system must be able to exchange information in real-time with the systems of many different companies. Messaging Platform can make this possible.

To illustrate some examples of Messaging Platform's uses, we describe:

- Accessing services on a remote system over the Web
- Making a Car.com service available for use by external systems
- Accessing a service provided by a proprietary ERP system

In reality, an organization can choose to limit Messaging Platform use to only one of the three scenarios, use two of the three scenarios, or implement all three scenarios. In other words, you can make your enterprise as open as possible or restrict inbound or outbound access.

Accessing Remote Services over the Web

Even though the insurance quotes come from various companies with many different systems, Car.com can interact with them because Car.com has agreed to use their XML schemas. These schemas define, among other things, the content and encoding requirements of a quote request and an actual quote. (For more information on schemas and their use, refer to the sidebar titled "What Are Schemas?")

Car.com customers can request and receive car insurance quotes over the Internet. To provide this service, Car.com uses Messaging Platform to access the quote services of various car insurance vendors.

1. Car.com implements the schema developed by the insurance company for the transmission of quote requests and quotes.
2. A user logs onto the Car.com website and requests a quote.
3. Car.com processes the data and creates a Quote Request object.
4. The Quote Request object is passed to Messaging Platform.
5. Messaging Platform transforms the Quote Request object into a (SOAP-encoded) XML document containing correctly encoded parameters as defied by the agreed-upon schema.
6. Messaging Platform sends the Quote Request XML document to the insurance company.
7. The insurance company receives the document and transforms it into a Quote Request that their system can use.
8. The insurance company processes the Quote Request and returns a Quote, formatted as an XML document. This is sent back to Car.com via Messaging Platform.

What Are Schemas?

A schema is a model for describing the structure of information. The term is borrowed from the database world to describe the structure of data in relational tables. In the context of XML, a schema describes a model for an entire class of documents. The model describes the possible arrangement of tags and text in a valid document. A schema might also be viewed as an agreement on a common vocabulary for a particular application that involves exchanging documents.

Schemas provide the ability to test the validity of XML documents sent and received over the Web. For example, if you're receiving XML transactions over the web, you don't want to process the content if it's not in the proper schema. Thus, schemas are proving very useful for the e-commerce environment, because they provide a mechanism to exchange accurate data between systems that would otherwise not be able to share. Many industries, such as insurance and telecommunications, are developing standardized schemas.

Schemas are similar to Datatype Definitions (DTD). Messaging Platform uses schemas instead of DTDs because schemas allow you to develop richer structural rules. In addition, there is vast and growing support in the computer industry for the use of XML schemas.

9. Messaging Platform receives the XML Quote document and verifies it against the schema. It then sends the appropriate data from the document to Car.com.
10. Car.com displays the Quote for its customer.

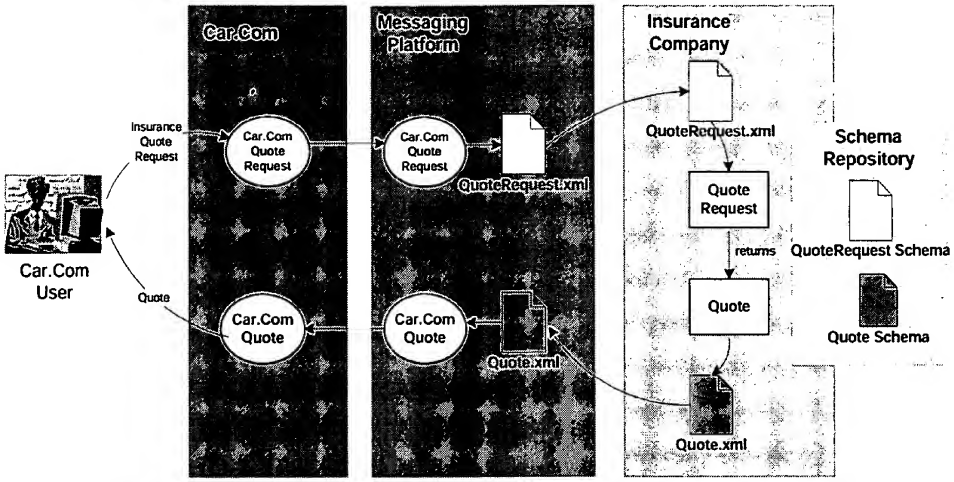


Figure 2. Using Messaging Platform to access services over the Web

Making Services Available over the Web

Car.com stores demographic information about its customers and makes it available for other vendors. Vendors can access information such as home address, age, and driving record and can then use this information to target people in specific age groups or locations for advertising purposes.

1. Car.com develops a schema for the demographic information that it makes available. Insurance vendors interested in purchasing the demographic data from Car.com agree to use this schema and make sure that they communicate with the demographic information services in accordance to the schema rules.

Messaging Platform and Schemas

The use of schemas is key to Messaging Platform. Their existence makes possible the exchange of data between disparate systems. Messaging Platform checks XML documents against the appropriate schema to ensure that the data is formatted correctly. It then culls out the data needed by the recipient to process the request. If the XML document contains parameters not needed by the recipient system, Messaging Platform simply ignores them. Later, Messaging Platform packages the return values for the sender, including all the parameters the sender packaged in the original XML document even though some of those fields were not touched by the recipient. Thus, Messaging Platform ties together two disparate applications, allowing them to use each others' services in spite of their differences.

2. Car.com's demographic service registers with the Messaging Platform Request Processor. Registering with the Request Processor makes the service available to receive data from external sources.
3. Insurance vendors send an XML document containing a request for demographic information to the Messaging Platform Request Processor.
4. The Request Processor forwards the XML document, sending all the data from the document to the Car.com demographic services.
5. The data from the request is processed by the demographic services, which return the requested information to the Request Processor.
6. The Request Processor repackages the returned information as an XML document that complies to the schema and sends it back to the requesting vendor. The vendor is then responsible for processing the XML document in its own way.

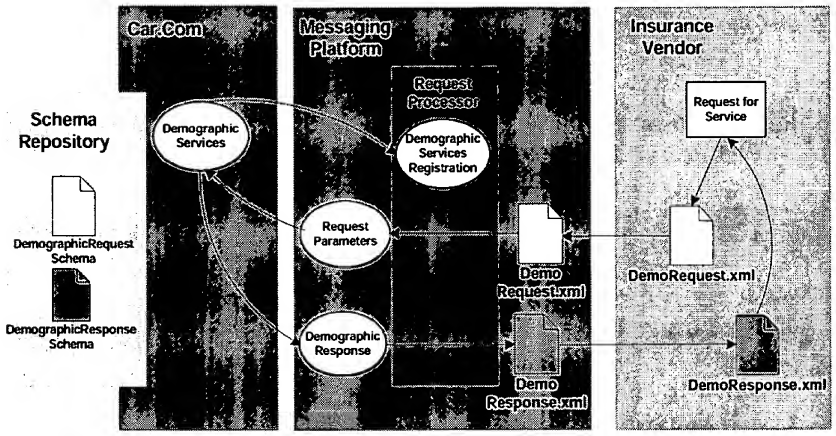


Figure 3. Using Messaging Platform to make services available over the Web

Using Proprietary Services over the Web

Large businesses, such as car manufacturers, often use Enterprise Relationship Planning (ERP) packages to control all aspects of their business, including inventory, billing, and shipping. Since Car.com allows users to order cars directly from manufacturers, Car.com must be able to exchange data with the manufacturer's ERP applications. Many ERPs are proprietary; adapters and connectors are required to link the ERPs with other applications. In this scenario, we describe how Car.com uses Messaging Platform to interact with an adapter and connector to access a car manufacturer's ERP.

1. A Car.com customer orders a car through Car.com. Among other things, he needs to find out when the car will be available for delivery.
2. Car.com processes the order and sends it to Messaging Platform for transmission to the car manufacturer.
3. Messaging Platform creates an XML Order document containing the purchase information. The XML document is formatted according to the rules of a schema published by the car manufacturer.
4. Messaging Platform sends the XML document to the adapter.
5. The adapter converts the information sent from Messaging Platform into a format that the connector can work with. (For example, it might convert the information into another XML document formatted differently than the one sent by Car.com.)
6. The adapter sends the request to a connector that has been developed specifically for the ERP. The connector parses the information in the request and sends it to the ERP, which then processes the order and returns requested information, such as a delivery date for the new car.
7. The connector sends the returned information to the adapter, which repackages the returned information as an XML Delivery Date document. Again, the XML document is formatted according to the manufacturer's schema.
8. The adapter sends the XML Delivery Date document to Messaging Platform. Messaging Platform processes the XML document and sends the Delivery Date data to Car.com.
9. Car.com processes the Delivery Date information and then displays the information for its customer.

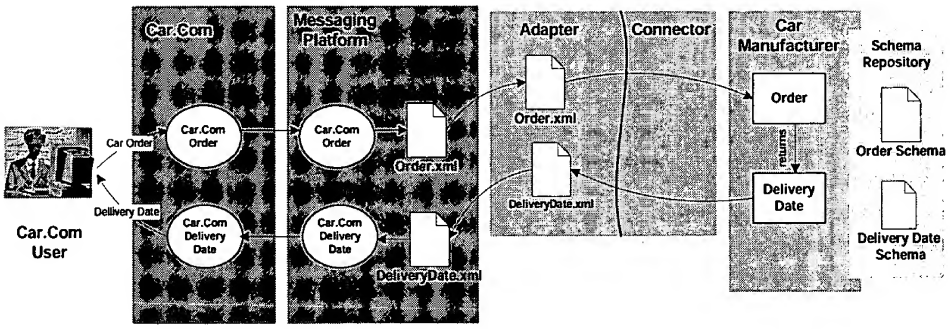


Figure 4. Using Messaging Platform to access proprietary services over the Web

Summary of Scenarios

The scenarios presented in this section describe a number of ways that Messaging Platform can be used within and across enterprises to connect and communicate between heterogeneous systems. In the scenarios, the sender and recipient remain quite decoupled, even when "connected" via Messaging Platform. This decoupling allows the sender and recipient to be constructed independently, glued together only by the agreement of a schema between them.

Messaging Platform provides a business object API to developers of the sender and recipient, shielding the intricate and error-prone construction of XML documents from these developers. This level of abstraction ensures a high level of productivity from a development team.

Tools, Components, and Frameworks

Messaging Platform provides functionality in four areas: connection management, protocol management, message structure, and transformation services. For each area, Messaging Platform provides various tools, components, and frameworks that simplify its use and implementation.

Connection Management. Messaging Platform provides a tool called Connection Assembler that developers use to create, manage, and manipulate Messaging Platform connections. These include connections from external systems to Messaging Platform and from Messaging Platform to external systems.

Protocol Management. Messaging Platform provides a framework for the implementation of Hypertext Transfer Protocol (HTTP) for transport. Although Messaging Platform uses HTTP as its default transfer protocol, open interfaces are provided to help developers implement the protocol of their choice.

Message Structure. Messaging Platform provides a tool called Schema Generator that developers can use to create XML documents containing the information to be transported to another system. Schema Generator also greatly simplifies the creation of the XML schema, which is a document used by systems to "read" the XML documents.

Messaging Platform also provides an Encoding Component that converts XML documents into encoded objects that can be understood and used by the system receiving them. Messaging Platform by default uses the Simple Object Access Protocol (SOAP) to encode XML documents as objects.

Transformation. Messaging Platform contains an Extensible Stylesheet Language (XSL) Translation Component that allows users to provide an alternative to using SOAP for encoding objects. For systems that do not use SOAP, XSL provides a way to transmit information in the format that the system understands.

Messaging Platform Tools and Components

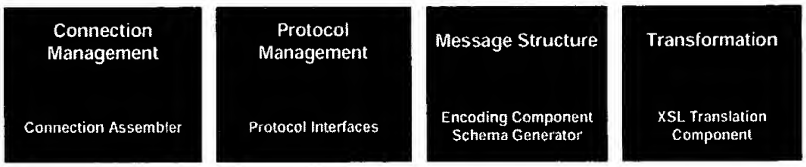


Figure 5. Messaging Platform tools, components, and frameworks

Messaging Platform Features

Messaging Platform's primary benefit is that it provides a way for disparate systems to communicate. This is accomplished through the use of supported and tested standards, such as XML, HTTP, and SOAP. Messaging Platform provides numerous other benefits:

Choice of transport and encoding mechanisms. By default, Messaging Platform uses some specific protocols, such as HTTP for message transport and SOAP for message encoding. However, users can adopt other protocols easily. For example, FTP could be used instead of HTTP and COINS or XMLRPC could be used instead of SOAP.

Flexible XML layout options. Messaging Platform allows users to determine the layout of their XML documents.

Choice of DOM implementation. A Document Object Model (DOM) determines how the information in an XML document is read and processed by the system. Numerous DOM implementations have been created by different companies and organizations. Messaging Platform provides Sun Microsystems' DOM by default; however, you can use any other DOM instead.

Choice of schema. Schemas are documents that describe the information contained in XML documents. Messaging Platform provides you with some default schemas, but does not limit you to their use. Messaging Platform supports any schema of your choice. This means that you can use a standard schema, a standard schema with some customization, or a schema that is unique.

Schema generation. Messaging Platform provides a GUI tool called the Schema Generator that automates the creation of schemas and the serialization of objects. The Schema Generator's GUI provides a simple and quick way to select the fields to be included in the schema and its serialized object, thus making it easy to modify schemas and serialized objects during the development process.

Object neutrality. Messaging Platform accommodates for differences in the way systems "think about" information. For example, a sender system might conceptualize an object that contains customer information as "Customer." However, the recipient system might consider that object to be "User." Despite their differences, these systems can exchange customer information because Messaging Platform transmits data in an object-neutral manner through the use of schemas. As long as the two systems agree to use the same schema, Customer and User objects can exchange data seamlessly.

Support for XSL. Because not everyone will want to encode objects using SOAP, Messaging Platform supports the use of XSL for formatting and presenting complex XML documents. Messaging Platform provides SAXON 4.5 as the XSL engine; however, you can replace it with the XSL processor of your choice.

Inbound Message Process

In Messaging Platform, the processes for receiving inbound messages and sending outbound messages differ. This section explains the process by which Messaging Platform receives inbound messages, or, more specifically, receives a request to use one of its services from an external system. Inbound messages are messages sent from a sender to a recipient via Messaging Platform.

The section starts with a high-level look at the inbound message process. It then provides more detail about how a recipient registers with the Request Processor and how Messaging Platform makes it possible for a sender to invoke a remote system's method.

Inbound Message Process Overview

This section provides a high-level overview of the Messaging Platform's inbound message process. During the inbound process, a sender sends a request to the recipient asking the recipient to run one of its methods and return the result to the sender.

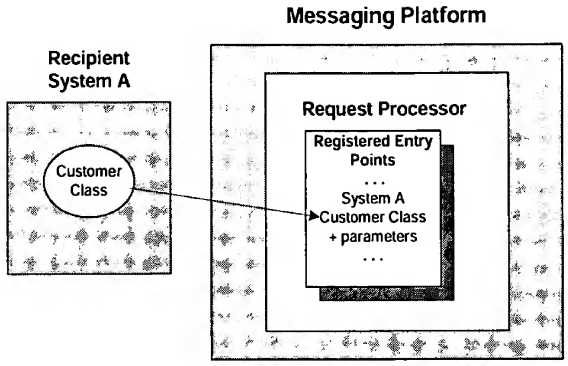
In this section we describe how:

- A recipient registers itself with the Messaging Platform Request Processor
- The sender sends a request to the Request Processor
- The Request Processor extracts information from the request and sends it to the recipient
- The Request Processor sends the recipient's return information back to the original sender

1. The recipient registers its "entry points" with the Request Processor.

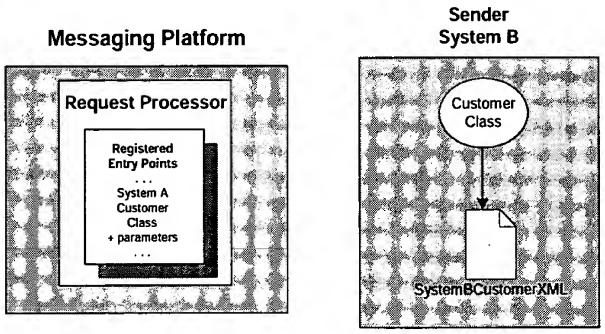
   Entry points are classes that are able to receive information from a remote sender. By registering its entry points, methods in the recipient are made available for calling via XML. Not every class in an application needs to be an entry point.

When a recipient registers, it tells the Request Processor about itself. For example, it registers its methods and the parameters the methods accept. The Request Processor saves this information for future reference.

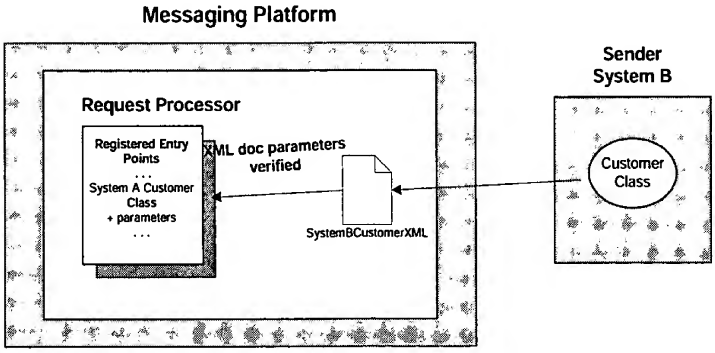


2. The sender creates an XML document containing a request to use one of the recipient's services.

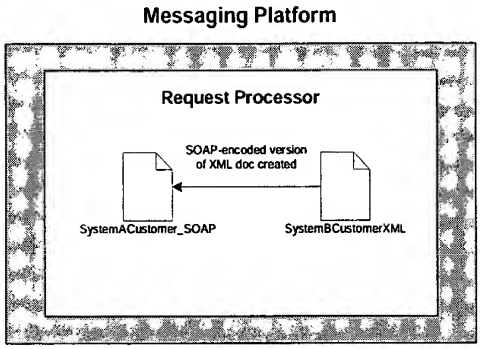
   For example, the sender's Customer class might need to get information about one of the recipient's customers. So, it sends a request to the recipient that will cause the recipient to run one of the methods it registered at the Request Processor to provide the needed information. The parameters to be passed to the method are also contained in the XML document.



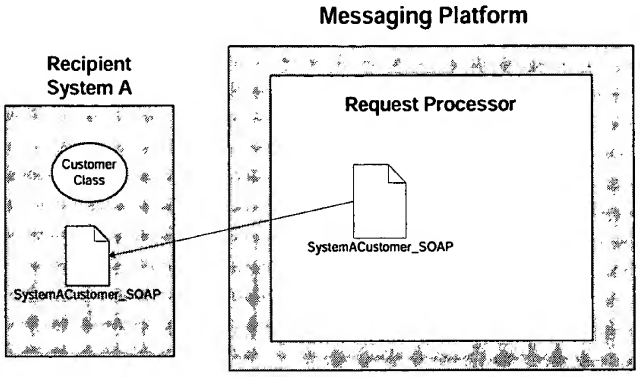
3. The sender sends an XML document to the Request Processor. Upon receiving the XML document, the Request Processor makes sure that the XML document can be accepted by its intended recipient.



4. The Request Processor identifies the method that the sender wishes to execute. The Request Processor then reviews the XML document, determines which parameters in the document are needed by the recipient, and creates SOAP-encoded versions of the parameters.



5. The Request Processor forwards the SOAP-encoded parameters to the recipient. These are recognized by the recipient as objects that implement the appropriate interface for each parameter.



6. The recipient method develops a response, or a return value. This value is sent to Request Processor, which encodes the return information for use by the sender in an XML document
7. Messaging Platform sends the XML document containing the response to the sender.

In the next two sections, "Registration" and "Invoking Methods at the Recipient," the steps presented above are explored in greater detail.

Registration

Although Messaging Platform provides some GUI tools, such as the SchemaGenerator and the ConnectionAssembler, for the most part it is a framework that simplifies the development and implementation of Messaging Platform. Because of this, the descriptions presented in this section and the next are largely descriptions of the functionality of various Messaging Platform classes and their objects.

Note: In this document, we are referring to classes in a generic manner. In reality, the Messaging Platform class names include information about the runtime being supported. For example, in Step 1 below, we refer to a RegistrationRequest object. In reality, this would be called the CPRegistrationRequest object if Component Platform were the runtime in use or EJBRegistrationRequest if the runtime were EJB.

1. The developer uses the Component Assembler to create a new Messaging Platform component. For more information on creating Messaging Platform components, go to Messaging Platform Component Templates on page 29.
2. Using the Launcher Component, the developer launches the component associated with the bean. When the component launches, the registration process is initiated.
3. As part of registration, the recipient's methods and parameters are packaged into a URL. The Request Processor receives the URL and validates it by making sure that all the content is present and in the correct form.
4. The Request Processor creates a RuntimeRequestHandler object for the registered component.

   The RuntimeRequestHandler contains a method called getInvokeable that uses reflection to obtain from the recipient class a complete list of its methods and their parameters. The RuntimeRequestHandler stores this list and the parameters contained in the registration URL for use later when it verifies that methods requested by senders are, in fact, methods that the recipient can run.

Note: There is a one-to-one relationship between registered components (recipients) and RuntimeRequestHandler objects.

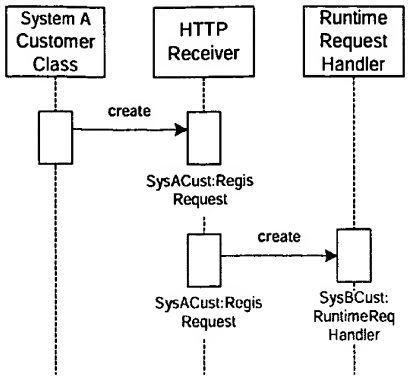


Figure 6. The recipient registration process

Invoking Methods at the Recipient

This section explains in detail the process by which a sender invokes a recipient's method. For clarity, we have separated it into three parts:

- Part One describes the process by which the sender creates an XML document and sends it to the Request Processor.
- Part Two describes the process by which the Request Processor repackages the XML document and sends it to the recipient.
- Part Three describes the process by which the recipient's return values are packaged and sent back to the sender.

Note: This description focuses on Component Platform for EJB.

Part One: From the Sender to the Request Processor

1. The sender uses the MessagingController factory class to create a new MethodCall object.

   The MethodCall object identifies the method to be invoked at the recipient and includes the parameters the sender wishes to pass to the method.

2. The sender uses MessagingController to create an Encoder object.

   The Encoder identifies the type of encoding required by the recipient.

3. The sender uses MessagingController to create a new RuntimeRequest object. The MethodCall and Encoder objects are passed to the new RuntimeRequest.
4. The sender tells RuntimeRequest to run the FireURL method. At this point, the RuntimeRequest object tells the Encoder to create a message containing the method and parameter information contained in the

MethodCall object. This message is encoded in the manner required by the recipient, which is specified by the Encoder object. (Since SOAP is the default Messaging Platform encoding mechanism, we will consider all encoded XML documents to be SOAP-encoded.)

5. The XML document is then attached to a URL, which the RuntimeRequest object sends to the Request Processor.

   The URL created in this step contains information that matches part of the URL sent to the Request Processor by the recipient during its registration process. The information that the recipient sent in the URL at registration is saved in the recipient's unique RuntimeRequestHandler, which is stored by the Request Processor.

   The URL contains the following information:
   http://server/mode-id/encoding-id/url + XML document.

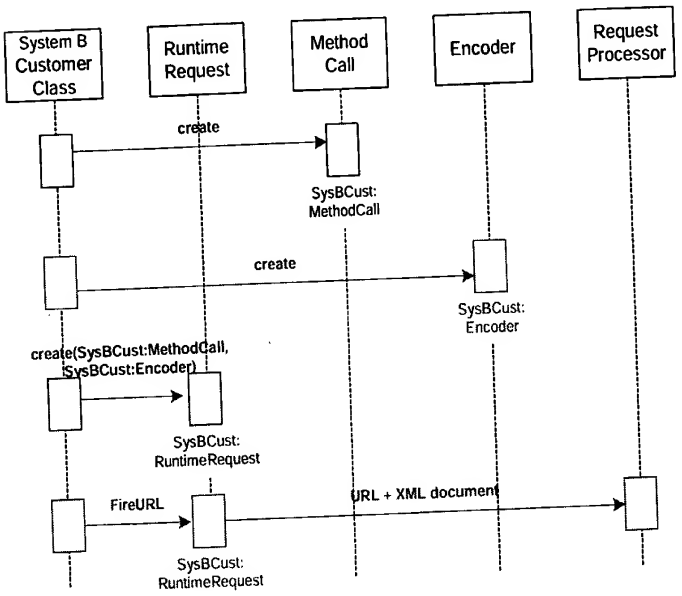


Figure 7. XML document is sent from the sender to the Request Processor

Part Two: From the Request Processor to the Recipient

1. The Request Processor receives the URL from the sender.
2. The Request Processor identifies the name of the RuntimeRequestHandler to be used for this request.

   The name of the RuntimeRequestHandler corresponds to the URL-ID contained in the URL.

3. The Request Processor determines the type of ContentHandler that must be created for the request. The type of ContentHandler needed is determined by the Encoding-ID contained in the URL. The Request Processor then makes a new ContentHandler object for the specified Encoding-ID.
4. When the XML document is received by the Request Processor from the network, the document is no more than a stream of bytes. This stream needs to be converted into an object that Messaging Platform can use. To do this, the Request Processor tells the ContentHandler to run its getContent method. This converts the stream of bytes into the appropriate object. For example, if SOAP is the encoding type, the appropriate object is a SOAP-encoded XML document.
5. The Request Processor tells the RuntimeRequestHandler to get its Invokeable and then sends the data contained in the XML document created in Step 4 to the Invokeable.
6. The Request Processor tells the Invokeable to run its Invoke method, which in fact runs the method and parameters specified in the data contained in the XML document.

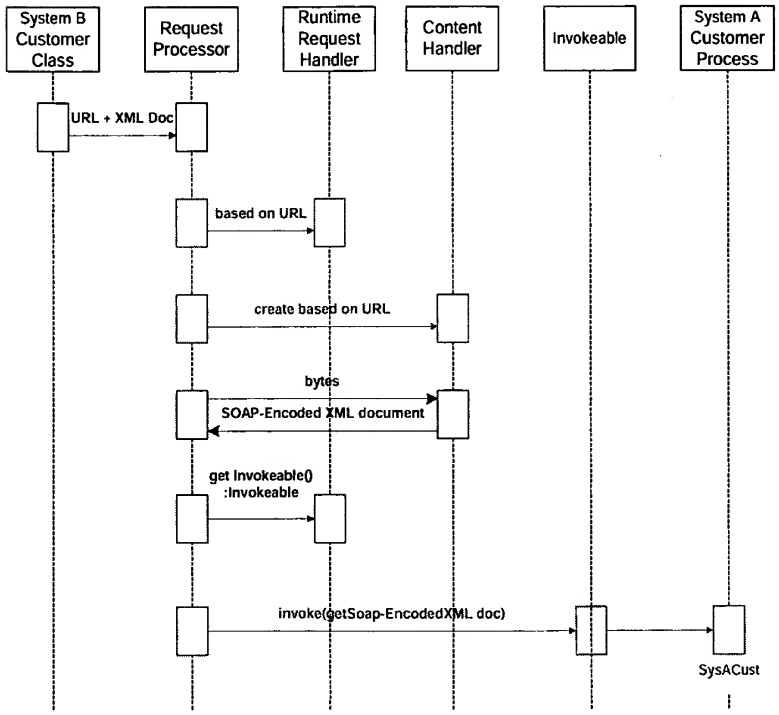


Figure 8. XML document is sent from the Request Processor to the recipient

Part Three: Returning Values to the Sender

When the invoked method runs, it returns a value. This value is returned to the sender via the Request Processor.

1. The invoked method may return a value, which is sent to the Request Processor.
2. The Request Processor sends the return value to the ContentHandler that was created when the original message was sent.
3. The ContentHandler contains a method called formResult, which takes the return value and returns it in the particular manner required by the sender (for example, SOAP-encoded).
4. The ContentHandler sends the returned object to the Request Processor.
5. The Request Processor sends the return value to the sender.

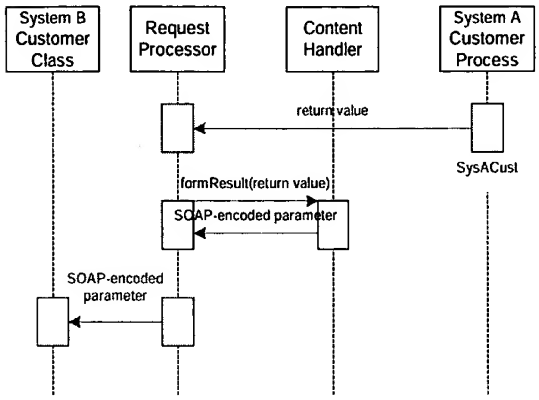


Figure 9. Return value is sent to sender

Outbound Message Process

This section explains Messaging Platform's process for sending messages from Component Platform for EJB to a recipient whose runtime is unknown. The outbound process allows you to send a request for services to an external system.

The outbound message process is considerably simpler than the inbound message process because of the Connection Assembler. The Connection Assembler is a GUI tool that automates the process of identifying and storing the addresses of remote destinations.

Note: This is the process for sending messages without the Connection Assembler. This document will be revised in the near future to include the Connection Assembler procedure instead.

Outbound Message Process

This section describes the process by which a sender sends information to a recipient that resides on a different system. It assumes that you are developing Component Platform components or other Java applications. The recipient can be any other system.

1. A connection information file is created for the recipient. This is an XML document that defines the:
   - Host on which the recipient resides (server)
   - Port on which the recipient resides
   - Mode-ID
   - Encoding-ID

   The name of this file becomes synonymous with the name of the connection.
2. The file is saved to [Component Repository]_Common\Messaging Runtime\Connections\Outbound directory.
3. A Connection object is created and passed the name of the connection that was specified in Step 1. (The name of the XML file is the name of the connection.)

4. A RuntimeRequest object is created and passed the recipient's URL and the method call to be invoked at the recipient.
5. The RuntimeRequest object runs the FireURL method. FireURL sends information encapsulated in the RuntimeRequest object to the Request Processor in a URL.
6. The sender may receive a return value from the recipient. If a return value is received, it is sent to the RuntimeRequest's getResponse method. RuntimeRequest creates an Encoder based on the type of encoding used by the returned document.

Messaging Platform Component Templates

This section briefly introduces Assembler templates and then provides you with information about each Messaging Platform templates. It explains what information you must provide for each template and what items the Assembler provides for you in return.

Assembler Templates

Component Platform provides a tool called the Component Assembler that you use to manage and configure components. The Assembler offers a number of predefined templates that you use to create new components. Templates provide you with a starting point for component creation. They provide you with the component's directory structure, default configurations, and some basic classes to which you add additional code to further develop the component. Since different templates create different types of components and runtime environments, the items provided by the Assembler differ for each template.

Five Messaging Platform components are available from the Assembler: CR Inbound, CR Outbound, J2EE Inbound, J2EE Outbound, and Adapter.

Note: For more information about Component Platform, refer to the *Component Platform Overview* guide.

For general information on using the Assembler, refer to the first two sections of *Developing WebLogic Applications with Component Platform* guide.

Component Runtime Inbound Template

The Component Runtime Inbound template creates a Component Runtime (CR) component that uses Messaging Platform to accept incoming requests. This component contains a service that can receive requests from external systems. It also contains other elements, such as a Request Processor and Launch Items, that are needed for inbound

messaging to be successful. Table 1 lists the parameters that you provide to the CR Inbound template to create a Messaging Platform Inbound Request component.

Table 1. Parameters Required by the CR Inbound Template

Parameter	Description
Request Processor Name	Name of the Request Processor.
Request Processor Port	The port on which the Request Processor listens.
Recipient Name	Name of the recipient. "Recipient" refers to the service that accepts requests from external systems.
Recipient Alias	The name by which Component Platform recognizes the recipient. This also is the recipient's URL address.
Recipient Type	A way to classify the recipient (for example, "CR Recipient"). The Explorer component uses this information to sort recipients.

The Assembler creates a new CR Inbound component from the information you provided. The component is a "skeleton" component that provides with you a solid starting point for further component development. Table 2 lists the items created for the new component.

Table 2 Items Created by the Assembler for CR Inbound Components

Item	Description
Request Processor process	A Component Platform server process that contains two named objects that you can run: the Request Processor and the Recipient.
Sender process	A Component Platform client process that submits requests to the Recipient through the Request Processor. This is provided for testing purposes.
Global, client, and server requirements	Other components or files required by the CR Inbound component.
Processor Launch Item	Launches the Request Processor and the Recipient.
Sender Launch Item	Launches the Sender GUI.
Source code	[Component Repository]\[Component Version]\src\[component_name]: \classes: [Recipient Name]Request.java = Packages a request as XML and then runs the FireURL method to send a request to the external service. In Inbound Messaging Platform processes, the Sender uses this class to package and send requests. \facilities: IDEStartup.java = Provides code to run compiled component \html InboundTest.html = Provides a Web page from which you can submit a request. Available for testing purposes.

Item	Description
	\inbound: Constants.java = Constants used by the component Note: The next three classes, working together, form a named object, specifically, a Recipient. This type of class grouping—an interface, a default implementation, and a stub—is used throughout Component Platform to create named objects. [Recipient]Recipient.java = Interface for the Recipient. [Recipient]RecipientImpl.java = Default implementation for the Recipient [Recipient]RecipientImpl_Stub.java = Stub for the Recipient Note: The next three classes, working together, form a Request Processor. [Request Processor]Processor.java = Interface for the Request Processor. [Request Processor]ProcessorImpl = Default implementation for the Request Processor. [Request Processor]ProcessorImpl_Stub.java = Stub for the Request Processor. \windows: [Recipient]InboundClientFrame.java = Creates a GUI for the Sender. This is available for testing purposes.

Further Developing the CR Inbound Component

The code provided by the Assembler, when compiled, provides you with enough material to run the component's processes. However, further development is, of course, required to add your desired functionality to the component.

Specifically, you will add code to the [Recipient]Recipient group of classes to create functionality. You do not need to make changes to any of the Request Processor classes; the Assembler creates a Request Processor that is fully functional.

Component Runtime Outbound Template

The Component Runtime Outbound template creates a Component Runtime (CR) component that uses Messaging Platform to send outgoing requests. This component contains a Sender process that the Sender uses to send a request to an external service. Table 3 lists the parameters that you provide to component.

Table 3. Parameters Required by the CR Outbound Template

Parameter	Description
Connection Name	Name of the Connection. Connection information is saved in an XML document that will also have this name.
Destination Host	The name of the host on which the Recipient resides.
Destination Port	The Recipient's port.
Destination URL	The Recipient's URL. This is also the name by which the Request Processor recognizes the Recipient; it must match the Recipient's alias provided when it was created using the Inbound template.
Destination Method	The Recipient method to be invoked by the request.
Destination Protocol	List of protocols - select the protocol used by the Recipient.
Destination Encoding	List of encoding options - select the encoding option used by the Recipient.

The Assembler creates a new CR Outbound component from the information you provided. The component is a "skeleton" component that provides with you a solid starting point for further component development. Table 4 lists the items created for the new component.

Table 4 Items Created by the Assembler for CR Outbound Components

Item	Description
Sender process	The process that sends request to an external service.
Global, client, and server requirements.	Other components or files required by the CR Outbound component.
Sender Launch Item	Launches the Sender GUI.
Source code	[Component Repository]\[Component Version]\src\[component_name]: \classes: [Component]Request.java = Packages a request as XML and then runs the FireURL method to send a request to the external service \facilities: IDEStartup.java =provides code to run compiled component \windows: [Sender]OneOutboundClientFrame.java = Creates a GUI for the Sender. This is available for testing purposes.

Further Developing the CR Outbound Component

The code provided by the Assembler, when compiled, provides you with enough material to run the component's processes. However, further development is, of course, required to add your desired functionality to the component.

You will probably extend the [Component]Request.java class so that it requests different or additional services. You may also extend the Sender, although it is more likely that you will create an entirely new Sender, using the one provided by the Assembler only for testing purposes.

Adapter Template

The Adapter template creates a Component Runtime component that uses Messaging Platform to send requests to and receive requests from a external system that is not using Messaging Platform. This component contains an Adapter process that listens for requests coming from external, proprietary systems. The Adapter does not register with a Request Processor; instead, it actually functions much as a request processor does, as it accepts requests and routes them to the correct service.

The component also contains a Sender process that the Sender uses to send a request to a proprietary system. Table 5 lists the parameters that you provide to the Adapter template to create a Messaging Platform Adapter component.

Table 5. Parameters Required by the Adapter Template

Parameter	Description
Adapter name	Name of the adapter. Connection information for the adapter in an XML document that will also have this name.
Port	Port on which adapter is listening for requests.

The Assembler creates a new Adapter component from the information you provided. The component is a "skeleton" component that provides with you a solid starting point for further component development. Table 6 lists the items created for the new component.

Table 6 Items Created by the Assembler for Adapter Components

Item	Description
Adapter process	The process that starts running the Adapter.
Sender process	The process that sends request to an external service via the Adapter.
Global, client, and server requirements.	Other components or files required by the Adapter component.
Adapter process Launch Item	Launches the Adapter.
Sender Launch Item	Launches the Sender GUI.
Source code	[Component Repository]\[Component Version]\src\[component_name]: \adapter: Constants.java = Constants used by the component [Adapter]AdapterConnectionImpl.java = Provides code for the Connector to which the Adapter sends requests Note: The next three classes, working together, form a named object, specifically, an Adapter. This type of class grouping—an interface, a default implementation, and a stub—is used throughout Component Platform to create named objects. [Adapter]Adapter.java = Interface for the Adapter [Adapter]AdapterImpl.java = Default implementation for the Adapter [Adapter]AdapterImpl_Stub.java = Stub for the Adapter \classes: [Component]Request.java = Packages a request as XML and then runs the FireURL method to send a request to the external service \facilities: IDEStartup.java =Provides code to run compiled component \html: AdapterTest.html = Provides a Web page from which you can submit a request that is sent via the Adapter. Available for testing purposes. \windows: [Adapter]OneClientFrame.java = Creates a GUI for the Sender. This is available for testing purposes.

Further Developing the Adapter Component

The code provided by the Assembler, when compiled, provides you with enough material to run the component's processes. However, further development is, of course, required to add your desired functionality to the component.

Specifically, you will add code to the [Adapter]AdapterConnectionImpl.java class. This class is, in fact, the Connection portion of the Adapter/Connector, and will need to be modified so that it can interact with an external system.

J2EE Inbound Template

The J2EE Inbound template creates an EJB component that uses Messaging Platform to accept incoming requests. This component contains a Bean that can receive requests from external systems. It also contains a process that allows you to monitor requests. Table 7 lists the parameters that you provide to the J2EE Inbound template to create a Messaging Platform Inbound Request component.

Table 7. Parameters Required by the J2EE Inbound Template

Parameter	Description
Home	Directory location of WebLogic.
Server Name	WebLogic server.
Server Host	Host on which the WebLogic server resides. The default value is localhost.
Server Port	Port on which the WebLogic server listens. The default value is 8001.
Initial Context Factory Class	The WebLogic initial context factory class. The default is weblogic.jndi.WLInitialContextFactory.
Provider URL	The server URL. The default is t3://localhost:8001
Security Principle	The server username. This value is optional.
Security Credential	The server password. This value is optional.
Bean Name	A name for the Bean to be created.
Bean Type	Currently, Messaging Platform supports only Stateless Beans.

The Assembler creates a new J2EE Inbound component from the information you provided. The component is a "skeleton" component that provides with you a solid starting point for further component development. Table 8 lists the items created for the new component.

Table 8 Items Created by the Assembler for J2EE Inbound Components

Item	Description
GUI Process	A Component Platform process that submits requests to the recipient. This is provided for testing purposes.
Monitor process	A Component Platform server process that provides a Receiver Monitor to monitor requests.
Server process	A WebLogic Runtime process deployment server process. This server launches its Beans and the Request Processor and then registers the Beans with the Request Processor
Global, client, and server requirements	Other components or files required by the J2EE component.
Inbound Test Console Launch Item	Launches the WebLogic Console.
Inbound Test GUI Launch Item	Launches the GUI Process.
Inbound Test Monitor Launch Item	Launches the Receiver Monitor as part of the Explorer Component.
Inbound Test Startup Launch Item	Launches the WebLogic deployment server.
Source code	[Component Repository]\[Component Version]\src\[component_name]: \beans: [bean].java = File created for the stateless Bean. This is a J2EE standard class. [bean]Bean.java = File created for the stateless Bean. This is a J2EE standard class. [bean]Home.java = File created for the stateless Bean. This is a J2EE standard class. \classes: Request.java = Packages a request as XML and then runs the FireURL method to send a request to the external service. In Inbound Messaging Platform processes, the Sender uses this class to package and send requests. \facilities: IDEStartup.java = Provides code to run compiled component \services: [server name]MonitorImpl.java = Creates a Receiver Monitor for the component. \windows: InboundClientFrame.java = Creates a GUI for the Sender. This is available for testing purposes.

Further Developing the J2EE Inbound Component

The code provided by the Assembler, when compiled, provides you with enough material to run the component's processes. However, further development is, of course, required to add your desired functionality to the component.

Specifically, you will add code to the Beans to create functionality. You will also update Request.java to map it to your Beans. You can also make changes to InboundClientFrame.java to improve its usefulness for your testing purposes. You do not need to make changes to MonitorImpl.java; the Assembler creates a Receiver Monitor that is fully functional.

J2EE Outbound Templates

The J2EE Outbound template creates a Component Runtime component that uses Messaging Platform to send outgoing requests. This component contains Beans that send requests to external services. Table 9 lists the parameters that you provide to component.

Table 9. Parameters Required by the J2EE Outbound Template

Parameter	Description
Home	Directory location of WebLogic.
Server Name	WebLogic server.
Server Host	Host on which the WebLogic server resides. The default value is localhost.
Server Port	Port on which the WebLogic server listens. The default value is 8001.
Initial Context Factory Class	The WebLogic initial context factory class. The default is weblogic.jndi.WLInitialContextFactory.
Provider URL	The server URL. The default is t3://localhost:8001
Security Principle	The server username. This value is optional.
Security Credential	The server password. This value is optional.
Destination Host	Host on which the recipient resides.
Destination Port	Port on which the recipient server listens.
Destination URL	The URL of the recipient server.
Destination Method	The method to be invoked at the recipient.
Destination Protocol	The protocol used by the recipient. Select the protocol from the list.
Destination Encoding	The encoding used by the recipient. Currently, SOAP is the only encoding option.
Bean Name	Name of the Bean.
Bean Type	Currently, only Stateless Beans can be created for MP.

The Assembler creates a new J2EE Outbound component from the information you provided. The component is a "skeleton" component that provides with you a solid starting point for further component development. Table 10 lists the items created for the new component.

Table 10. Items Created by the Assembler for J2EE Outbound Components

Item	Description
Server process	A WebLogic Runtime process. This is a deployment server that launches the Beans.
Monitor process	A Component Platform server process that provides a Receiver Monitor to monitor requests.
Global, client, and server requirements.	Other components or files required by the J2EE component.
Outbound Test Console Launch Item	Launches the WebLogic Console.
Outbound Test Monitor	Launches the Receiver Monitor as part of the Explorer Component.
Outbound Test Startup	Launches the WebLogic deployment server.
Source code	[Component Repository]\[Component Version]\src\[component_name]: \beans: [bean].java = File created for the stateless Bean. This is a J2EE standard class. [bean]Bean.java = File created for the stateless Bean. This is a J2EE standard class. [bean]Home.java = File created for the stateless Bean. This is a J2EE standard class. \classes: Request.java = Packages a request as XML and then runs the FireURL method to send a request to the external service \services: [server name]MonitorImpl.java = Creates a Receiver Monitor for the component.

Further Developing the J2EE Outbound Component

The code provided by the Assembler, when compiled, provides you with enough material to run the component's processes. However, further development is, of course, required to add your desired functionality to the component.

Specifically, you will add code to the Beans to create functionality. You might also need to update Request.java. You do not need to make changes to MonitorImpl.java; the Assembler creates a Receiver Monitor that is fully functional.

Monitoring Tools

This section describes the Messaging Platform's monitoring tools. Currently, Messaging Platform provides a tool that allows you to monitor the Request Processor. In the future, additional monitoring tools will be available.

Monitoring the Request Processor

Two tools are available to help you monitor incoming requests received by the Request Processor: the GUI Receiver Monitor and the XML Monitor. Both tools report the number of requests received at the Request Processor and the number of responses it returned to the sender; the monitors also indicates the size of each message, in bytes.

The two monitoring tools provide you with the same information. The GUI Receiver Monitor is a Component Platform component. The Receiver Monitor supports Component Runtime and Enterprise Java Bean runtimes. It can be customized to support other runtimes as well. The XML tool can be viewed from Microsoft's Internet Explorer, version 5.0. The XML tool provides a way to monitor the Request Processor from a remote location.

Using the Request Processor Monitoring Tools

This section describes, at a high level, how to access and use the Request Processor monitoring tools.

Note: Both descriptions assume that you are using a WebLogic server with Messaging Platform. The actual procedure with your application server may differ slightly.

The Receiver Monitor

To use the Receiver Monitor (when using WebLogic):

1. Start the ObjectSpace.
2. Start the WebLogic server.
3. Start the Component Explorer.
4. Start the Receiver Monitor.

5. Run the application that will be receiving messages. Make sure that the application registers its services with the Request Processor, and then initiate a transaction. The viewer displays the request history.

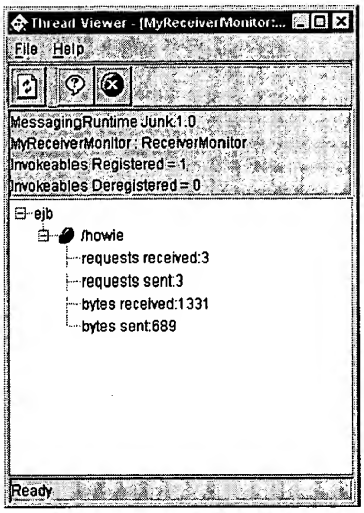


Figure 10. The Receiver Monitor

In Figure 10, only one service (invokeable) is registered at the request processor. If multiple services were registered, the Receiver Monitor would display each service and their track requests.

The XML Monitor

The XML Monitor provides the same information as the Receiver Monitor. When you use the XML Monitor, you must know the invokeable's URL and enter it in the browser's address line. The XML Monitor then takes over, reporting on Request Processor's activity.

Note: The XML Monitor requires Internet Explorer 5.0; it will not work with other Web browsers.

The URL has the following format:

http://[host]:[port]/[mode-ID]/admin

To use the XML Monitor:

1. Start the ObjectSpace.
2. Start the WebLogic server.
3. Run the application that will be receiving messages. Make sure that the application registers its services with the Request Processor, and then initiate a transaction. The viewer displays the request history.
4. Using IE 5.0, enter the URL for the service registered at the Request Processor using MP for the Mode-ID (for example: http://localhost:8001/mp/admin).

Information similar to the following appears in the browser window:

```
<?xml version="1.0" ?>
- <mpadmin>
    <registered>3</registered>
    <deregistered>0</deregistered>
  - <modelist>
      <mode>ejb</mode>
    </modelist>
  </mpadmin?
```

When you use MP as the Mode-ID, you receive general information about Request Processor activity. For example, you see that requests have been registered at the request processor and that they are using EJB services. However, this view does not tell you whether or not the requests have been answered, or how large the requests are.

5. For additional information about Request Processor activity, enter the same URL you did in step 4, but substitute the actual Mode-ID for MP (for example, EJB or CP).

   A report similar to the following appears in the browser:

```
<?xml version="1.0" ?>
- <modeadmin name="ejb">
  - <invokeable url="/howie">
    <bytessent>1</bytessent?
    <bytesrecieved>1</bytesreceived>
    <requestssent>1</requestssent>
    <requestsreceived>1</requestsreceived>
  </invokeable>
</modeadmin>
```

Appendix A—Schema and DTD for Connection Documents

Appendix A presents the schema and document type definition (DTD) for connection files that you create as part of the configuration process.

Schema for Connection Documents

```
<?xml version="1.0"?>
<!-- XML Schema for Messaging Platform Connection files -->
<schema xmlns="http://www.xpedior.com/mp/ConnectionSchema"
        targetNS="http://www.xpedior.comp/mp/ConnectionSchema"
        version="0.7.5">

<element name="host" type="string" />
<element name="port" type="positive-integer" />
<element name="protocolid" type="string" />
<element name="encodingid" type="string" />
```

DTD for Connection Documents

```
<!-- DTD for Messaging Platform Connection files -->
<!DOCTYPE connection [
  <!ELEMENT host (#PCDATA)>
  <!ELEMENT port (#PCDATA)>
  <!ELEMENT protocolid (#PCDATA)>
  <!ELEMENT encodingid (#PCDATA)>
]>
```

Glossary

ActiveX

A loosely defined set of technologies developed by Microsoft. ActiveX is an outgrowth of two other Microsoft technologies called OLE (Object Linking and Embedding) and COM (Component Object Model).

applet

A program designed to be executed from within another application in the same operating system process. Unlike a applications, applets cannot be executed directly from the operating system. Note that applets are not limited to Java applets running in Web browsers.

bean

See Java Bean

component

A small binary object or program that performs a specific function and is designed in such a way to easily operate with other components and applications. Each Component Platform component is uniquely identified by a version number.

Document Object Model (DOM)

A World Wide Web Consortium (W3C) specification that describes the structure of dynamic HTML and XML documents as a logical structure instead of a collection of tagged words. A DOM defines a document as a tree-like hierarchy of nodes in which the document is an object containing other objects, such as images or forms. Through the DOM API, programs and scripts can access these objects.

Extensible Markup Language (XML)

A condensed form of Standard Generalized Markup Language (SGML). It allows for the creation of customized tags, enabling the definition, transmission, validation, and interpretation of data between applications and between organizations. This differs from HTML tags, which simply specify the format of information.

Extensible Stylesheet Language (XSL)

A language for specifying stylesheets that apply formatting to complex XML documents for presentation in HTML or other formats. Unlike cascading style sheets (CSS), which maps an XML element to a single display object, XSL can map a single XML element to more than one type of display object.

Hypertext Transfer Protocol (HTTP)

A protocol used to request and transmit files, especially Web pages and Web page components, over the Internet or other computer network.

interface

Need a good definition for this

JavaBean

A specification developed by Sun Microsystems that defines how Java objects interact. An object that conforms to this specification is called a JavaBean, and is similar to an ActiveX component. It can be used by any application that understands the JavaBeans format.

The principal difference between ActiveX controls and JavaBeans is that ActiveX controls can be developed in any programming language but executed only on a Windows platform, whereas JavaBeans can be developed only in Java, but can run on any platform.

protocol

An standard agreement for transmitting data between two devices. The protocol determines the type of error checking to be used, the data compression method, if any, how the sending device will indicate that it has finished sending a message, and how the receiving device will indicate that it has received a message.

reflection

A feature of the Java programming language that allows an executing Java program to examine, or "introspect," upon itself, and manipulate internal properties of the program. For example, it's possible for a Java class to obtain the names of all its methods and parameters.

runtime

The base layer of software that enables applications to execute.

schema

A model for describing the structure of information.

serialize

In object-oriented programming, to convert an object into a stream of bytes that can be written out to a file or a disk and later reconstituted into an object.

servlet

An applet that runs on a server. Servlets run within the JVM of the Sun Microsystems Servlet Runner or another server-enabled Web server.

signature

The parameters, such as name and parameters, that are unique to a method and by which it can be identified.

Simple Object Access Protocol (SOAP)

A Remote Procedure Call (RPC) mechanism that is used to transport information across a network. SOAP uses HTTP as the transport mechanism and uses XML documents to encode objects.

XML

See Extensible Markup Language

XSL

See Extensible Stylesheet Language

Index